Eric Hopkins, Dillion Klobe, Brandon Becker

Computer Engineering 3150

10 December 2014

Project Two: Switch/Keyboard Problem

The objective of this project is to design a self-contained 8051-based device (Simon2 Board) which will be able to do keyboard and music applications. The simon2 board can be programmed “in place” using a serial port connection. Also, this board has 9 push-buttons, 9 LEDs, and a speaker, most of which were utilized in our project. Other hardware also includes the use of a breadboard to connect a seven segment display to the Simon2 board. The programs utilized included FlashMagic programming software, Keil microvision for compiling the code, and GitHub to collaborate between group members.

We choose to tackle the keyboard problem because of Eric’s expertise in music. Our board plays two short tunes stored in program memory through the speaker named Star Wars, and The Sorcerer’s Apprentice. These tunes are stored in code memory as 3 arrays each: an array of ints with the frequency of each note, an array of chars with the duration in 18ths of a second, and an array of chars holding the name of the song.

The required delays for the desired frequencies were calculated as follows.

If we check with a frequency of 500 Hz, we have 7373 cycles, which is clearly within the range mode 1 of the timer is capable of, as will any higher frequency. Additionally, to be careful to avoid potential overflow of a constant, instead of using 1843250, we approximate this with 61425 \* 30, and perform the division by f before doing the multiplication, to ensure our value remains within the bounds of int (this approximation is slightly low, which will be compensated for somewhat by the time it takes to reset the clock and complement the speaker between waves). This calculated value is then subtracted from 65536, and the high and low bytes are set to the high and low bytes for timer 0.

The duration given in 18ths of a second is much simpler. Since Hz is simply cycles per second, we can multiple by the duration in seconds to get how many cycles of the wave we need to perform.

Contrary to the frequencies and durations, the delay functions used were more of a trial, then we calculated the amount of time we ended up with. For example, our long delay between changing modes and other button presses, we decided we liked the amount of time from running the full timer in mode 1 (65536 cycles) 12 times. This was later calculated to be

The smaller delay functions were approached in a similar manner, using this known 0.21s delay as a guideline for the duration of our other delays.

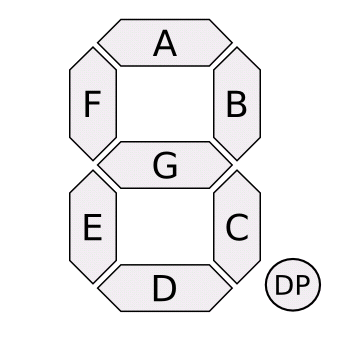
As Eric’s added feature, he used timer 1 with interrupts to make LED patterns while these songs are playing. He ran into some issues setting up the interrupts properly, mostly to avoid messing up the serial communication function and it’s required interrupts, so he had to be sure to enable and disable only the timer 1 interrupt using clever use of the bitwise and and or operators. An added complication was making sure there was enough time between changes of the pattern (to avoid potentially causing epileptic seizures like the first working attempt could have). This was solved through clever use of static member variables, thus only changing the pattern once every 12 times the interrupt was actually tripped (approximately 0.21s as shown above).

The song title is also displayed on the PC screen using serial communication. We encountered much difficulty in getting the serial communication to work properly, such as the baud rate being set incorrectly in the uart\_init() function and interrupts not being enabled at the proper times. With these errors, anything from no output on the computer to the board not even performing other basic functions was occurring, though this cleared up once the serial transmission issues were fixed.

The next required feature was the piano capabilities of the board. Each button has a distinctive output on the seven segment display and a distinctive pitch. Issues such as absurd frequencies or improper frequency calculation were encountered, but easily remedied. Other than this, we had no significant problems in this section of the project.

Dillion’s and Brandon’s added features work hand-in-hand together. Dillion introduced a third mode of operation in which the seven segment display and a couple of switches are used as a counter (from 0 to F in hex) with rollover, in a style similar to project 1. To complement this feature, Brandon worked on using this counter to add two numbers (within the range of the counter) and make a unique sound (similar to Mario collecting a coin) when the sum overflows the counter space. These two features’ biggest issue came from the delay caused while updating the seven segment display. Initially, while waiting for button presses to operate the counter/adder, the function was repeatedly checking 20 conditional statements, so any button had to be held for a prolonged period of time for the desired action to be performed. Fixing this involved refining the cases in which the display was updated, making it only update when necessary, instead of constantly updating. Additionally, some complications arose when outputting to the display at any point, with many ports which were necessary to avoid, leading to use of some switch and LED ports as outputs to the display, as shown in the table below.

|  |  |
| --- | --- |
| Port Pins | Seven Segment Display Pins |
| P2^5 | a |
| P2^4 | b |
| P2^3 | c |
| P0^0 | d |
| P1^4 | e |
| P0^3 | f |
| P1^6 | g |



In this project, we used many of the features available in C and on the microcontroller to improve our project. We used timers for generating all of our delays and frequencies. Additionally, interrupts alongside timers for the LED designs and serial communication with a PC through a COM port were used. Some advanced programming techniques like static variables and enums were also used to improve readability and functionality of code.

The work was distributed according to the preferences and capabilities of the group members. As such, Eric was the primary programmer, while Dillion and Brandon did more with the hardware aspects of wiring and such. However, much of this was done while working together, and final decisions were a product of the group, as was the project report.